

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE	3. REPORT TYPE AND DATES COVERED Final 22 Apr 90 to 22 Apr 91
4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report: Computer Sciences Corporation, MC Ada V1.2.beta/ Concurrent Computer Corporation, Concurrent/Masscomp 5600 (Host) to Concurrent/Masscomp 5600 (Target), 900121S1.10251			5. FUNDING NUMBERS
6. AUTHOR(S) National Institute of Standards and Technology Gaithersburg, MD USA			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Institute of Standards and Technology National Computer Systems Laboratory Bldg. 255, Rm. A266 Gaithersburg, MD 20899 USA			
8. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ada Joint Program Office United States Department of Defense Washington, D.C. 20301-3081			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) Computer Science Corporation, Gaithersburg MD, Concurrent/Masscomp 5600 under Unix; Masscomp RTU V4.1 (Host) to Concurrent/Masscomp (Dual 68020 processor configuration under CSC developed Ada Real-Time Operating System (ARTOS) for bare machine environments (Target), ACVE 1.10.			
14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office			15. NUMBER OF PAGES
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED			16. PRICE CODE
18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED		19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT

AVF Control Number: NIST89CSC545_1.10
18 March 1990
23 April 1990

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 900121S1.10251
Computer Sciences Corporation
MC Ada V1.2.beta/ Concurrent Computer Corporation
Concurrent/Masscomp 5600 Host and
Concurrent/Masscomp 5600 (Dual 68020 processor configuration) Target

Completion of On-Site Testing:
21 January 1990
22 April 1990

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899



Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Application For	
1. <input checked="checked" type="checkbox"/> Ada Compiler	
2. <input type="checkbox"/> Ada Library	
3. <input type="checkbox"/> Ada Preprocessor	
4. <input type="checkbox"/> Ada Debugger	
5. <input type="checkbox"/> Ada Test Environment	
6. <input type="checkbox"/> Ada Development Environment	
7. <input type="checkbox"/> Ada Documentation	
8. <input type="checkbox"/> Ada Source Code	
9. <input type="checkbox"/> Ada Object Code	
10. <input type="checkbox"/> Ada Executable	
A-1	

Ada Compiler Validation Summary Report:

Compiler Name: MC Ada V1.2.beta/ Concurrent Computer
Corporation

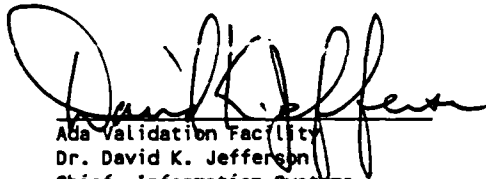
Certificate Number: 900121S1.10251

Host: Concurrent/Masscomp 5600 under Unix; Masscomp RTU V4.1

Target: Concurrent/Masscomp 5600 (Dual 68020 processor configuration) under
CSC developed Ada Real-Time Operating System (ARTOS) for bare machine
environments

Testing Completed 22 April 1990 Using ACVC 1.10

This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	IMPLEMENTATION CHARACTERISTICS	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS	3-6
3.7	ADDITIONAL TESTING INFORMATION	3-6
3.7.1	Prevalidation	3-6
3.7.2	Test Method	3-7
3.7.3	Test Site	3-7
APPENDIX A	CONFORMANCE STATEMENT	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	
APPENDIX E	COMPILER OPTIONS AS SUPPLIED BY Computer Sciences Corporation	

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A.

This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

(K0) (—

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by Gemma Corp. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 22 April 1990 at Computer Sciences Corporation.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language

constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage.

Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: MC Ada V1.2.beta/ Concurrent Computer Corporation

ACVC Version: 1.10

Certificate Number: 900121S1.10251

Host Computer:

Machine: Concurrent/Masscomp 5600

Operating System: Unix; Masscomp RTU V4.1

Memory Size: 4MB

Target Computer:

Machine: Concurrent/Masscomp 5600 (Dual 68020 processor configuration)

Operating System: CSC developed Ada Real-Time Operating System (ARTOS)
for bare machine environments

Memory Size: 4MB

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

- (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)
- (3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)
- (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

- (1) This implementation supports the additional predefined types `SHORT_INTEGER`, `SHORT_FLOAT`, and `TINY_INTEGER` in the package `STANDARD`. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- (1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- (2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)
- (3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)
- (4) `NUMERIC_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- (5) `NUMERIC_ERROR` is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)
- (6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- (1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)
- (2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)
- (3) The method used for rounding to integer in static universal real expressions is round to even. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

- (1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises no exception. (See test C36003A.)
- (2) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components. (See test C36202A.)
- (3) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `SYSTEM.MAX_INT + 2` components. (See test C36202B.)
- (4) A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array

type is declared. (See test C52103X.)

- (5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array objects are declared. (See test C52104V.)
- (6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)
- (7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)
- (8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

- (1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

- (1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)
- (2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- (3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

- (1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

- (1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)
- (2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)
- (3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)
- (4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- (5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)
- (6) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

- (7) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)
- (8) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

j. Input and output.

- (1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
- (2) The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)
- (3) The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 549 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 242 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 8 tests was required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	129	1132	1784	17	16	46	3124
Inapplicable	0	6	531	0	12	0	549
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER														TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14		
Passed	198	577	545	245	172	99	161	331	137	36	252	292	79	3124	
Inapplicable	14	72	135	3	0	0	5	1	0	0	0	77	242	549	
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44	
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717	

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD5011D	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B	E28005C	ED7004B	ED7005C	ED7005D
ED7006C	ED7006D				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 549 tests were inapplicable for the reasons indicated:

- a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

- b. The following 2 tests are inapplicable because this implementation does not support predefined type `LONG_FLOAT`.

C35702B B86001U

- c. The following 8 tests are inapplicable because this implementation does not support a 48 bit integer machine size.

C45531M	C45531N	C45531O	C45531P
C45532M	C45532N	C45532O	C45532P

- d. The following 16 tests are inapplicable because this implementation does not support a predefined type LONG_INTEGER.

B52004D	B55b09C	B86001W	C45231C
C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C
C45631C	C45632C	C55b07A	CD7101F

- e. The following (1) test is inapplicable because this implementation does not support predefined fixed point types other than DURATION.

B86001Y

- f. The following (1) test is inapplicable because this implementation does not support floating point type with a name other than FLOAT or SHORT_FLOAT.

B86001Z

- g. The following (1) test is inapplicable because this implementation does not support recompilation of package SYSTEM invalidating other ARTOS packages that require package SYSTEM.

C86001F

- h. The following (1) test is inapplicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

C96005B

- i. The following 14 tests are inapplicable because this implementation does not support size clauses for floating point types.

CD1009C	CD2A41A	CD2A41B	CD2A41E
CD2A42A	CD2A42B	CD2A42C	CD2A42D
CD2A42E	CD2A42F	CD2A42G	CD2A42H
CD2A42I	CD2A42J		

- j. The following 2 tests are inapplicable because this implementation does not support size specifications for array types that imply compression of component type when the component type is a composite or floating point type. This implementation requires an explicit size clause on the component type.

CD2A61I CD2A61J

- k. The following 10 tests are inapplicable because this implementation does not support size clauses for access types. Access types are represented by machine addresses which are 32 bits.

CD2A84B	CD2A84C	CD2A84D	CD2A84E
CD2A84F	CD2A84G	CD2A84H	CD2A84I
CD2A84K	CD2A84L		

- l. The following 5 tests are inapplicable because this implementation does not support size clauses for task types.

CD2A91A	CD2A91B	CD2A91C	CD2A91D
CD2A91E			

- m. The following 42 tests are inapplicable because this implementation does not support 'address clauses where a dynamic address is applied to a variable requiring an initialization. The AVO has ruled that these tests may be declared inapplicable.

CD5003B..H (7 tests)	CD5011A..H (8 tests)
CD5011L..N (3 tests)	CD5011Q..R (2 tests)
CD5012A..I (9 tests)	CD5012L CD5013B
CD5013D CD5013F	CD5013H CD5013L
CD5013N CD5013R	CD5014T..X (5 tests)

- n. The following 3 tests are inapplicable because this implementation does not support address clauses for tasks.

CD5012J	CD5013S	CD5014S
---------	---------	---------

- o. The following 242 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102A..C (3 tests)	CE2102G..H (2 tests)
CE2102K	CE2102N..Y (12 tests)
CE2103C..D (2 tests)	CE2104A..D (4 tests)
CE2105A..B (2 tests)	CE2106A..B (2 tests)
CE2107A..H (8 tests)	CE2107L
CE2108A..B (2 tests)	CE2108C..H (6 tests)
CE2109A..C (3 tests)	CE2110A..D (4 tests)
CE2111A..I (9 tests)	CE2115A..B (2 tests)
CE2201A..C (3 tests)	CE2201F..N (9 tests)
EE2201D..E (2 tests)	EE2401D EE2401G
CE2204A..D (4 tests)	CE2205A
CE2208B	CE2401A..C (3 tests)
CE2401E..F (2 tests)	CE2401H..L (5 tests)
CE2404A..B (2 tests)	CE2405B
CE2406A	CE2407A..B (2 tests)
CE2408A..B (2 tests)	CE2409A..B (2 tests)
CE2410A..B (2 tests)	CE2411A
CE3102A..B (2 tests)	EE3102C
CE3102F..H (3 tests)	CE3102J..K (2 tests)
CE3103A	CE3104A..C (3 tests)
CE3107B	CE3108A..B (2 tests)
CE3109A	CE3110A
CE3111A..B (2 tests)	CE3111D..E (2 tests)
CE3112A..D (4 tests)	
CE3114A..B (2 tests)	CE3115A
EE3203A	CE3208A
EE3301B	CE3302A
CE3305A	CE3402A
EE3402B	CE3402C..D (2 tests)
CE3403A..C (3 tests)	CE3403E..F (2 tests)
CE3404B..D (3 tests)	CE3405A
EE3405B	CE3405C..D (2 tests)
CE3406A..D (4 tests)	CE3407A..C (3 tests)
CE3408A..C (3 tests)	CE3409A
CE3409C..E (3 tests)	EE3409F
CE3410A	CE3410C..E (3 tests)
EE3410F	CE3411A CE3411C

CE3412A	EE3412C
CE3413A	CE3413C
CE3602A..D (4 tests)	CE3603A
CE3604A..B (2 tests)	CE3605A..E (5 tests)
CE3606A..B (2 tests)	CE3704A..F (6 tests)
CE3704M..O (3 tests)	CE3706D
CE3706F..G (2 tests)	CE3804A..P (16 tests)
CE3805A..B (2 tests)	CE3806A..B (2 tests)
CE3806D..E (2 tests)	CE3806G..H (2 tests)
CE3905A..C (3 tests)	CE3905L
CE3906A..C (3 tests)	CE3906E..F (2 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 8 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B24009A	B26005A	B33301B	B38003A
B38003B	B38009A	B38009B	B41202A

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the Computer Sciences Corporation compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the MC Ada V1.2.beta/ Concurrent Computer Corporation compiler and the CSC developed Ada Real-Time Operating System (RTOS) for bare machine environments using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The testing was completed in two steps. The first step included witness testing of all tests except the execution of the I/O tests. The second step was the execution of the I/O tests on 22 April 1990. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	Concurrent/Masscomp 5600
Host operating system:	Unix; Masscomp RTU V4.1
Target computer:	Concurrent/Masscomp 5600 (Dual 68020 processor configuration)
Target operating system:	CSC developed Ada Real-Time Operating System (RTOS) for bare machine environments

A tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precision was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were not customized before being written to the tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the tape.

TEST INFORMATION

The contents of the tape were loaded onto a VAX/780 disk storage medium using the VAX backup/recover command. The files were then transmitted to the Masscomp 5600 (M5600) over two serial lines using 'Kermit'. The server 'Kermit' at the M5600 end stored the files on the disk.

The executable tests were run on the target computer with the CSC developed Ada Real-Time Operating System (ARTOS) for bare machine environments running in the single processing and multi-processing modes. All results were captured using the 'Kermit' log session command executing with the VAX/780 as a terminal emulator. The result files were then printed using the VAX/780.

The compiler was tested using command scripts provided by Computer Sciences Corporation and reviewed by the validation team. See Appendix E for a complete listing of the compiler options for this implementation. The compiler options invoked during this test were:

-E and -w

Test output, compilation listings, and job logs were captured on tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Computer Sciences Corporation and was completed on 22 April 1990.

APPENDIX A

DECLARATION OF CONFORMANCE

Computer Sciences Corporation has submitted the following Declaration of Conformance concerning the MC Ada V1.2.beta/ Concurrent Computer Corporation.

APPENDIX A

DECLARATION OF CONFORMANCE

Customer: Computer Sciences Corporation
Integrated Systems Division
304 West Route 38
Moorestown, NJ 08057

Ada Validation Facility: National Institute of Standards and
Technology

ACVC Version: V1.10

Ada Implementation:

Ada Compiler: MC Ada V1.2.beta/Concurrent Computer Corporation
Version: V1.2.beta

Host Computer System: Concurrent/Masscomp 5600

Host Operating System: UNIX; Masscomp RTU V4.1

Target Computer System: Concurrent/Masscomp 5600 (Dual 68020
processor configuration)

Target Operating System: CSC developed Ada Real-Time Operating System
(ARTOS) for bare machine environments

Operating System Version: V1.0

Customers's Declaration:

We, the undersigned, representing Computer Sciences Corporation (CSC) declare that CSC has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration. We declare that CSC is the owner of record of the Ada Real-Time Operating System listed above. CSC has been licensed as a user of the compiler listed above, and under this license has implemented the Ada Run Time System. Concurrent Computer Corporation retains ownership of this Ada Language Compiler. Computer Sciences Corporation requests that the Ada Validation Certificate specifically identifying ARTOS as the Run Time Environment be issued in CSC's name. CSC assumes all legal responsibility for having the validation certificate issued in CSC's name.


Ralph M. Mattei, Center Chief Scientist

4/20/90
Date


Peter M. Cahn, Center Vice President

4/20/90
Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the MC Ada V1.2.beta/ Concurrent Computer Corporation compiler and CSC developed Ada Real-Time Operating System (ARTOS) for bare machine environments, as described in this Appendix, are provided by Computer Sciences Corporation. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

...

type TINY_INTEGER is range -128..127;

type INTEGER is range -2147483648..2147483647;

type SHORT_INTEGER is range -32768..32767;

type FLOAT is digits 15 range
-1.79769313486232E308..1.79769313486232E308;

type SHORT_FLOAT is digits 6 range -3.40282E38..3.40282E38;

type DURATION is delta 1.0E-3 range
-2.147483648E6..2.147483648E6

...

end STANDARD;

ATTACHMENT I

APPENDIX F. Implementation-Dependent Characteristics

1. Implementation-Dependent Pragmas

1.1. `INLINE_ONLY` Pragma

The `INLINE_ONLY` pragma, when used in the same way as pragma `INLINE`, indicates to the compiler that the subprogram must always be inlined. This pragma also suppresses the generation of a callable version of the routine which save code space.

1.2. `BUILT_IN` Pragma

The `BUILT_IN` pragmas is used in the implementation of some predefined Ada packages, but provides no user access. It is used only to implement code bodies for which no actual Ada body can be provided, for example the `MACHINE_CODE` package.

1.3. `SHARE_CODE` Pragma

The `SHARE_CODE` pragma takes the name of a generic instantiation or a generic unit as the first argument and one of the identifiers `TRUE` or `FALSE` as the second argument. This pragma is only allowed immediately at the place of a declarative item in a declarative part or package specification, or after a library unit in a compilation, but before any subsequent compilation unit.

When the first argument is a generic unit the pragma to all instantiations of that generic. When the first argument is the name of a generic instantiation the pragma applies only to the specified instantiation, or overloaded instantiations.

If the second argument is `TRUE`, the compiler will try to share code generated for other instantiations of the same generic. When the second argument is `FALSE` each instantiation will get a unique copy of the generated code. The extent to which code is shared between instantiations depends on this pragma and the kind of generic formal parameters declared for the generic unit.

The name pragma `SHARE_BODY` is also recognized by the implementation and has the same effect as `SHARE_CODE`. It is included for compatibility with earlier versions of VADS.

1.4. `NO_IMAGE` Pragma

The pragma suppresses the generation of the image array used for the IMAGE attribute of enumeration types. This eliminates the overhead required to store the array in the executable image.

1.5. EXTERNAL_NAME Pragma

The EXTERNAL_NAME pragma takes the name of a subprogram or variable defined in Ada and allows the user to specify a different external name that may be used to reference the entity from other languages. The pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification.

1.6. INTERFACE_NAME Pragma

The INTERFACE_NAME pragma takes the name of a variable defined in another language and allows it to be referenced directly in Ada. The pragma will replace all occurrences of the variable name with an external reference to the second, link_argument. The pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification. The object must be declared as a scalar or an access type. The object cannot be any of the following:

- a loop variable,
- a constant,
- an initialized variable,
- an array, or
- a record.

1.7. IMPLICIT_CODE Pragma

Takes one of the identifiers ON or OFF as the single argument. This pragma is only allowed within a machine code procedure. It specifies that implicit code generated by the compiler be allowed or disallowed. A warning is issued if OFF is used in any implicit code needs to be generated. The default is ON.

2. Implementation of Predefined Pragmas

2.1. CONTROLLED

This pragma is recognized by the implementation but has no effect.

2.2. ELABORATE

This pragma is implemented as described in Appendix B of the Ada RM.

2.3. INLINE

The pragma is implemented as described in Appendix B of the Ada RM.

2.4. INTERFACE

This pragma supports calls to 'C' and FORTRAN functions. The Ada subprograms can be either functions or procedures. The types of parameters and the result type for functions must be scalar, access or the predefined type ADDRESS in SYSTEM. Record and array objects can be passed by reference using the ADDRESS attribute.

2.5. LIST

This pragma is implemented as described in Appendix B of the Ada RM.

2.6. MEMORY_SIZE

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

2.7. OPTIMIZE

This pragma is recognized by the implementation but has no effect.

2.8. PACK

This pragma will cause the compiler to choose a non-aligned representation for composite types. It will not cause objects to be packed at the bit level.

2.9. PAGE

This pragma is implemented as described in Appendix B of the Ada RM.

2.10. PRIORITY

This pragma is implemented as described in Appendix B of the Ada RM.

2.11. SHARED

This pragma is recognized by the implementation but has no effect.

2.12. STORAGE_UNIT

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

2.13. SUPPRESS

This pragma is implemented as described, except that RANGE_CHECK and DIVISION_CHECK cannot be suppressed.

2.14. SYSTEM_NAME

This pragma is recognized by the implementation. The implementation does not allow SYSTEM to be modified by means of pragmas, the SYSTEM package must be recompiled.

3. Implementation-Dependent Attributes

3.1. P'REF

For a prefix that denotes an object, a program unit, a label, or an entry:

This attribute denotes the effective address of the first of the storage units allocated to P. For a subprogram, package, task unit, or label, it refers to the address of the machine code associated with the corresponding body or statement. For an entry for which an address clause has been given, it refers to the corresponding hardware interrupt. The attribute is of the type OPERAND defined in the package MACHINE_CODE. The attribute is only allowed within a machine code procedure.

See section F.4.8 for more information on the use of this attribute.

(For a package, task unit, or entry, the 'REF attribute is not supported.)

4. Specification of Package SYSTEM

```
package SYSTEM
is
  type NAME is ( masscomp_unix );

  SYSTEM_NAME      : constant NAME := masscomp_unix;

  STORAGE_UNIT     : constant := 8;
  MEMORY_SIZE      : constant := 16_777_216;

  -- System-Dependent Named Numbers

  MIN_INT          : constant := -2_147_483_648;
  MAX_INT          : constant := 2_147_483_647;
  MAX_DIGITS       : constant := 15;
  MAX_MANTISSA     : constant := 31;
```

```

FINE_DELTA      : constant := 2.0*(-31);
TICK            : constant := 0.0166666;

-- Other System-dependent Declarations

subtype PRIORITY is INTEGER range 0 .. 99;

MAX_REC_SIZE : integer := 64*1024;

type ADDRESS is private;

NO_ADDR : constant ADDRESS;

function PHYSICAL_ADDRESS(I: INTEGER) return ADDRESS;
function ADDR_GT(A, B: ADDRESS) return BOOLEAN;
function ADDR_LT(A, B: ADDRESS) return BOOLEAN;
function ADDR_GE(A, B: ADDRESS) return BOOLEAN;
function ADDR_LE(A, B: ADDRESS) return BOOLEAN;
function ADDR_DIFF(A, B: ADDRESS) return INTEGER;
function INCR_ADDR(A: ADDRESS; INCR: INTEGER) return ADDRESS;
function DECR_ADDR(A: ADDRESS; DECR: INTEGER) return ADDRESS;

function ">"(A, B: ADDRESS) return BOOLEAN renames ADDR_GT;
function "<"(A, B: ADDRESS) return BOOLEAN renames ADDR_LT;
function ">="(A, B: ADDRESS) return BOOLEAN renames ADDR_GE;
function "<="(A, B: ADDRESS) return BOOLEAN renames ADDR_LE;
function "-"(A, B: ADDRESS) return INTEGER renames ADDR_DIFF;
function "+"(A: ADDRESS; INCR: INTEGER) return ADDRESS renames
INCR_ADDR;
function "--"(A: ADDRESS; DECR: INTEGER) return ADDRESS renames
DECR_ADDR;

pragma inline(PHYSICAL_ADDRESS);
pragma inline(ADDR_GT);
pragma inline(ADDR_LT);
pragma inline(ADDR_GE);
pragma inline(ADDR_LE);
pragma inline(ADDR_DIFF);
pragma inline(INCR_ADDR);
pragma inline(DECR_ADDR);

private

type ADDRESS is new integer;

NO_ADDR : constant ADDRESS := 0;

end SYSTEM;

```

5. Restrictions On Representation Clauses

5.1. Pragma PACK

In the absence of pragma PACK record components are padded so as to provide for efficient access by the target hardware, pragma PACK applied to a record eliminate the padding where possible. Pragma PACK has no other effect on the storage allocated for record components a record representation is required.

5.2. Record Representation Clauses

For scalar types a representation clause will pack to the number of bits required to represent the range of the subtype. A record representation applied to a composite type will not cause the object to be packed to fit in the space required. An explicit representation clause must be given for the component type. An error will be issued if there is insufficient space allocated.

5.3. Address Clauses

Address clauses are supported for variables and constants.

5.4. Interrupts

Interrupt entries are not supported.

5.5. Representation Attributes

The ADDRESS attribute is not supported for the following entities:

- Packages
- Tasks
- Labels
- Entries

5.6. Machine Code Insertions

Machine code insertions are supported.

The general definition of the package MACHINE_CODE provides an assembly language interface for the target machine. It provides the necessary record type(s) needed in the code statement, an enumeration type of all the operation code mnemonics, a set of register definitions, and a set of addressing mode functions.

The general syntax of a machine code statement is as follows:

```
CODE_n'(operation_code, operand(,operand));
```

where n indicated the number of operands in the aggregate.

A special case arises for a variable number of operands. The operands are listed within a subaggregate. The format is as follows:

```
CODE_N'(operation_code,(operand(,operand)));
```

For those operation codes that require no operands, named notation must be used (cf.RM4.3(4)).

```
CODE_0'(op=>operation_code);
```

The operation code must be an enumeration literal (i.e. it cannot be an object, attribute, or a rename).

An operand can only be entity defined in MACHINE_CODE or the 'REF attribute.

The arguments to any of the functions defined in MACHINE_CODE must be static expressions, string literals, or the functions defined in MACHINE_CODE. The 'REF attribute may not be used as an argument in any of these functions.

Inline expansion of machine code procedures is supported.

6. Conventions for Implementation-generated Names

There are no implementation-generated names.

7. Interpretation of Expressions in Address Clauses

Address clauses are supported for constants and variables.

8. Restrictions on Unchecked Conversions

None.

9. Restrictions on Unchecked Deallocations

None.

10. Implementation Characteristics of I/O Packages

The ARTOS does not currently support DIRECT_IO or SEQUENTIAL_IO, but it provides its own unique I/O packages that allow real-time synchronous and asynchronous input and output. The package TEXT_IO is supported, however, the only valid files are STANDARD_INPUT and STANDARD_OUTPUT. Use of TEXT_IO functions or procedures that operate

on other external files causes the USE_ERROR exception to be raised at execution time.

11. Implementation Limits

The following limits are actually enforced by the implementation. It is not intended to imply that resources up to or even near these limits are available to every program.

11.1. Line Length

The implementation supports a maximum line length of 500 characters including the end of line character.

11.2. Record and Array Sizes

The maximum size of a statically sized array type is 4,000,000 x STORAGE_UNITS. The maximum size of a statically sized record type is 4,000,000 x STORAGE_UNITS. A record type or array type declaration that exceeds these limits will generate a warning message.

11.3. Default Stack Size for Tasks

In the absence of an explicit STORAGE_SIZE length specification every task except the main program is allocated a fixed size stack of 10,240 STORAGE_UNITS. This is the value returned by T'SORAGE_SIZE for a task type T.

11.4. Default Collection Size

In the absence of an explicit STORAGE_SIZE length attribute the default collection size for an access type is 100 times the size of the designated type. This is the value returned by T'SORAGE_SIZE for an access type T.

11.5. Limit on Declared Objects

There is an absolute limit of 6,000,000 x STORAGE_UNITS for objects declared statically within a compilation unit. If this value is exceeded the compiler will terminate the compilation of the unit with a FATAL error message.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

```

-- MACRO.DFS                      -- ACVC_VERSION_1.10
-- Modified for the MC-Ada Version 1.2 validated 18-Aug-1989
-- Modified for use with CSC Ada Real-Time Executive by A. Preston
-- 16-Oct-1989
-- THIS FILE CONTAINS THE MACRO DEFINITIONS USED IN THE ACVC TESTS.
-- THESE DEFINITIONS ARE USED BY THE ACVC TEST PRE-PROCESSOR,
-- MACROSUB. MACROSUB WILL CALCULATE VALUES FOR THOSE MACRO SYMBOLS
-- WHOSE DEFINITIONS DEPEND ON THE VALUE OF MAX_IN_LEN (NAMELY, THE
-- VALUES OF THE MACRO SYMBOLS BIG_ID1, BIG_ID2, BIG_ID3, BIG_ID4,
-- BIG_STRING1, BIG_STRING2, MAX_STRING_LITERAL, BIG_INT_LIT,
-- BIG_REAL_LIT, MAX_LEN_INT_BASED_LITERAL, MAX_LEN_REAL_BASED_LITERAL,
-- AND BLANKS). THEREFORE, ANY VALUES GIVEN IN THIS FILE FOR THOSE
-- MACRO SYMBOLS WILL BE IGNORED BY MACROSUB.

-- NOTE: THE MACROSUB PROGRAM EXPECTS THE FIRST MACRO IN THIS FILE TO
--       BE MAX_IN_LEN.-- FOR OUR CLIENTS_SAMPLE_TESTS_ACVC_VERSION_1.10

-- EACH DEFINITION IS ACCORDING TO THE FOLLOWING FORMAT:

-- A. A NUMBER OF LINES PRECEDED BY THE ADA COMMENT DELIMITER, --.
--    THE FIRST OF THESE LINES CONTAINS THE MACRO SYMBOL AS IT APPEARS
--    IN THE TEST FILES (WITH THE DOLLAR SIGN). THE NEXT FEW "COMMENT"
--    LINES CONTAIN A DESCRIPTION OF THE VALUE TO BE SUBSTITUTED.
--    THE REMAINING "COMMENT" LINES, THE FIRST OF WHICH BEGINS WITH THE
--    WORDS "USED IN: " (NO QUOTES), CONTAIN A LIST OF THE TEST FILES
--    (WITHOUT THE .TST EXTENSION) IN WHICH THE MACRO SYMBOL APPEARS.
--    EACH TEST FILE NAME IS PRECEDED BY ONE OR MORE BLANKS.
-- B. THE IDENTIFIER (WITHOUT THE DOLLAR SIGN) OF THE MACRO SYMBOL,
--    FOLLOWED BY A SPACE OR TAB, FOLLOWED BY THE VALUE TO BE
--    SUBSTITUTED. IN THE DISTRIBUTION FILE, A SAMPLE VALUE IS
--    PROVIDED; THIS VALUE MUST BE REPLACED BY A VALUE APPROPRIATE TO
--    THE IMPLEMENTATION.

-- DEFINITIONS ARE SEPARATED BY ONE OR MORE EMPTY LINES.
-- THE LIST OF DEFINITIONS BEGINS AFTER THE FOLLOWING EMPTY LINE.

-- $MAX_IN_LEN
-- AN INTEGER LITERAL GIVING THE MAXIMUM LENGTH PERMITTED BY THE
-- COMPILER FOR A LINE OF ADA SOURCE CODE (NOT INCLUDING AN END-OF-LINE
-- CHARACTER).
-- USED IN: A26007A
MAX_IN_LEN      499

-- $BIG_ID1
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- THE MACROSUB PROGRAM WILL SUPPLY AN IDENTIFIER IN WHICH THE
-- LAST CHARACTER IS '1' AND ALL OTHERS ARE 'A'.
-- USED IN: C23003A C23003B C23003C B23003D B23003E C23003G
--          C23003H C23003I C23003J C35502D C35502F

```

```

BIG_ID1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA

-- $BIG_ID2
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID1 ONLY IN THE LAST CHARACTER.  THE MACROSUB
-- PROGRAM WILL USE '2' AS THE LAST CHARACTER.
-- USED IN: C23003A C23003B C23003C B23003F C23003G C23003H
--          C23003I C23003J
BIG_ID2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA

-- $BIG_ID3
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- MACROSUB WILL USE '3' AS THE "MIDDLE" CHARACTER; ALL OTHERS ARE 'A'.
-- USED IN: C23003A C23003B C23003C C23003G C23003H C23003I
--          C23003J
BIG_ID3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A3AAAAA

-- $BIG_ID4
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID3 ONLY IN THE MIDDLE CHARACTER.  MACROSUB
-- WILL USE '4' AS THE MIDDLE CHARACTER.
-- USED IN: C23003A C23003B C23003C C23003G C23003H C23003I
--          C23003J
BIG_ID4
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A4AAAAA

-- $BIG_STRING1
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING2
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN: C35502D C35502F
BIG_STRING1
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA

```



```

-- $BIG_STRING2
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING1
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN: C35502D C35502F
BIG_STRING2
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA1"

```

```

-- $MAX_STRING_LITERAL
-- A STRING LITERAL CONSISTING OF $MAX_IN_LEN CHARACTERS (INCLUDING THE
-- QUOTE CHARACTERS).
-- USED IN: A26007A
MAX_STRING_LITERAL
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```

```

-- $NEG_BASED_INT
-- A BASED_INTEGER LITERAL (PREFERABLY BASE 8 OR 16) WHOSE HIGHEST ORDER
-- NON-ZERO BIT WOULD FALL IN THE SIGN BIT POSITION OF THE
-- REPRESENTATION FOR SYSTEM.MAX_INT, I.E., AN ATTEMPT TO WRITE A
-- NEGATIVE VALUED LITERAL SUCH AS -2 BY TAKING ADVANTAGE OF THE
-- BIT REPRESENTATION.
-- USED IN: E24201A
NEG_BASED_INT 16#FFFFFFFD#

```

```

-- $BIG_INT_LIT
-- AN INTEGER LITERAL WHOSE VALUE IS 298, BUT WHICH HAS
-- ($MAX_IN_LEN - 3) LEADING ZEROES.
-- USED IN: C24003A
BIG_INT_LIT
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
000

```

```

-- $BIG_REAL_LIT
-- A UNIVERSAL_REAL LITERAL WHOSE VALUE IS 690.0, BUT WHICH HAS
-- ($MAX_IN_LEN - 5) LEADING ZEROES.
-- USED IN: C24003B C24003C
BIG_REAL_LIT
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
00

```

```

-- $MAX_LEN_INT_BASED_LITERAL
-- A BASED_INTEGER LITERAL (USING COLONS) WHOSE VALUE IS 2:11:, HAVING
-- ($MAX_IN_LEN - 5) ZEROES BETWEEN THE FIRST COLON AND THE FIRST 1.
-- USED IN: C2A009A
MAX_LEN_INT_BASED_LITERAL
2:0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
000000000000

```

```

-- $MAX_LEN_REAL_BASED_LITERAL
-- A BASED_REAL LITERAL (USING COLONS) WHOSE VALUE IS 16:F.E:, HAVING
-- ($MAX_IN_LEN - 7) ZEROES BETWEEN THE FIRST COLON AND THE F.

```

```
-- $BLANKS
-- A SEQUENCE OF ($MAX_IN_LEN - 20) BLANKS.
-- USED IN:  B22001A  B22001B  B22001C  B22001D  B22001E  B22001F
--           B22001G  B22001H  B22001I  B22001J  B22001K  B22001L  B22001M
--           B22001N
--           <          LIMITS OF SAMPLE SHOWN BY ANGLE BRACKETS          >
BLANKS
```

```
-- $MAX_DIGITS
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_DIGITS.
-- USED IN:  B35701A  C07102B
MAX_DIGITS      15

-- $NAME
-- THE NAME OF A PREDEFINED INTEGER TYPE OTHER THAN INTEGER,
-- SHORT_INTEGER, OR LONG_INTEGER.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS  NO_SUCH_TYPE_AVAILABLE.)
-- USED IN:  AVAT007  C45231D  B86001X  C7D101G
NAME TINY_INTEGER
```

```
-- $FLOAT_NAME
-- THE NAME OF A PREDEFINED FLOATING POINT TYPE OTHER THAN FLOAT,
-- SHORT_FLOAT, OR LONG_FLOAT. (IMPLEMENTATIONS WHICH HAVE NO SUCH
-- TYPES SHOULD USE AN UNDEFINED IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT013  B86001Z
FLOAT_NAME      NO_SUCH_TYPE
```

```
-- $FIXED_NAME
-- THE NAME OF A PREDEFINED FIXED POINT TYPE OTHER THAN DURATION.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN: AVAT015 B86001Y
FIXED_NAME      NO_SUCH_FIXED_TYPE
```

```
-- $INTEGER_FIRST
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS INTEGER'FIRST.
-- THE LITERAL MUST NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503F  B54801B
INTEGER FIRST -2 147 483 648
```

```
-- $INTEGER_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST. THE LITERAL MUST
-- NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING BLANKS.
-- USED IN: C35503F C45232A B45B01B
INTEGER LAST 2 147 483 647
```

```
-- $INTEGER_LAST_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST + 1.
-- USED IN: C45232A
INTEGER LAST PLUS_1 2_147 483 648
```

```

-- $MIN_INT
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS SYSTEM.MIN_INT.
-- THE LITERAL MUST NOT CONTAIN UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN: C35503D C35503F CD7101B
MIN_INT -2147483648

-- $MAX_INT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT.
-- THE LITERAL MUST NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN: C35503D C35503F C4A007A CD7101B
MAX_INT 2147483647

-- $MAX_INT_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT + 1.
-- USED IN: C45232A
MAX_INT_PLUS_1 2_147_483_648

-- $LESS_THAN_DURATION
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE (NOT SUBJECT TO
-- ROUND-OFF ERROR IF POSSIBLE) LIES BETWEEN DURATION'BASE'FIRST AND
-- DURATION'FIRST. IF NO SUCH VALUES EXIST, USE A VALUE IN
-- DURATION'RANGE.
-- USED IN: C96005B
LESS_THAN_DURATION -100_000.0

-- $GREATER_THAN_DURATION
-- A REAL LITERAL WHOSE VALUE (NOT SUBJECT TO ROUND-OFF ERROR
-- IF POSSIBLE) LIES BETWEEN DURATION'BASE'LAST AND DURATION'LAST. IF
-- NO SUCH VALUES EXIST, USE A VALUE IN DURATION'RANGE.
-- USED IN: C96005B
GREATER_THAN_DURATION 100_000.0

-- $LESS_THAN_DURATION_BASE_FIRST
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE IS LESS THAN
-- DURATION'BASE'FIRST.
-- USED IN: C96005C
LESS_THAN_DURATION_BASE_FIRST -10_000_000

-- $GREATER_THAN_DURATION_BASE_LAST
-- A REAL LITERAL WHOSE VALUE IS GREATER THAN DURATION'BASE'LAST.
-- USED IN: C96005C
GREATER_THAN_DURATION_BASE_LAST 10_000_000

-- $COUNT_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.COUNT'LAST.
-- USED IN: CE3002B
COUNT_LAST 2_147_483_647

-- $FIELD_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.FIELD'LAST.
-- USED IN: CE3002C
FIELD_LAST 2_147_483_647

-- $ILLEGAL_EXTERNAL_FILE_NAME1
-- AN ILLEGAL EXTERNAL FILE NAME (E.G., TOO LONG, CONTAINING INVALID
-- CHARACTERS, CONTAINING WILD-CARD CHARACTERS, OR SPECIFYING A
-- NONEXISTENT DIRECTORY).
-- USED IN: CE2103A CE2102C CE2102H CE2103B CE3102B CE3107A
ILLEGAL_EXTERNAL_FILE_NAME1 /illegal/file_name/2/2[]$X2102C.DAT

-- $ILLEGAL_EXTERNAL_FILE_NAME2

```

```

-- AN ILLEGAL EXTERNAL FILE NAME, DIFFERENT FROM $EXTERNAL_FILE_NAME1.
-- USED IN: CE2102C CE2102H CE2103A CE2103B
ILLEGAL_EXTERNAL_FILE_NAME2 /illegal/file_name/CE2102C*.DAT

-- $ACC_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE MINIMUM NUMBER OF BITS
-- SUFFICIENT TO HOLD ANY VALUE OF AN ACCESS TYPE.
-- USED IN: CD2A81A CD2A81B CD2A81C CD2A81D CD2A81E
--          CD2A81F CD2A81G CD2A83A CD2A83B CD2A83C CD2A83E
--          CD2A83F CD2A83G ED2A86A CD2A87A
ACC_SIZE 32

-- $TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS A SINGLE ENTRY WITH ONE INOUT PARAMETER.
-- USED IN: CD2A91A CD2A91B CD2A91C CD2A91D CD2A91E
TASK_SIZE 32

-- $MIN_TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS NO ENTRIES, NO DECLARATIONS, AND "NULL;"
-- AS THE ONLY STATEMENT IN ITS BODY.
-- USED IN: CD2A95A
MIN_TASK_SIZE 32

-- $NAME_LIST
-- A LIST OF THE ENUMERATION LITERALS IN THE TYPE SYSTEM.NAME, SEPARATED
-- BY COMMAS.
-- USED IN: CD7003A
NAME_LIST MASSCOMP_UNIX

-- $DEFAULT_SYS_NAME
-- THE VALUE OF THE CONSTANT SYSTEM.SYSTEM_NAME.
-- USED IN: CD7004A CD7004C CD7004D
DEFAULT_SYS_NAME MASSCOMP_UNIX

-- $NEW_SYS_NAME
-- A VALUE OF THE TYPE SYSTEM.NAME, OTHER THAN $DEFAULT_SYS_NAME. IF
-- THERE IS ONLY ONE VALUE OF THE TYPE, THEN USE THAT VALUE.
-- NOTE: IF THERE ARE MORE THAN TWO VALUES OF THE TYPE, THEN THE
-- PERTINENT TESTS ARE TO BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN: ED7004B1 CD7004C
NEW_SYS_NAME MASSCOMP_UNIX

-- $DEFAULT_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.STORAGE_UNIT.
-- USED IN: CD7005B ED7005D3M CD7005E
DEFAULT_STOR_UNIT 8

-- $NEW_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA STORAGE_UNIT, OTHER THAN $DEFAULT_STOR_UNIT. IF THERE
-- IS NO OTHER PERMITTED VALUE, THEN USE THE VALUE OF
-- $SYSTEM.STORAGE_UNIT. IF THERE IS MORE THAN ONE ALTERNATIVE,
-- THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN: ED7005C1 ED7005D1 CD7005E
NEW_STOR_UNIT 8

-- $DEFAULT_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MEMORY_SIZE.
-- USED IN: CD7006B ED7006D3M CD7006E
DEFAULT_MEM_SIZE 16_777_216

-- $NEW_MEM_SIZE

```

```

-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA MEMORY_SIZE, OTHER THAN $DEFAULT_MEM_SIZE. IF THERE IS NO
-- OTHER VALUE, THEN USE $DEFAULT_MEM_SIZE. IF THERE IS MORE THAN
-- ONE ALTERNATIVE, THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR
-- EACH ALTERNATIVE. IF THE NUMBER OF PERMITTED VALUES IS LARGE, THEN
-- SEVERAL VALUES SHOULD BE USED, COVERING A WIDE RANGE OF
-- POSSIBILITIES.
-- USED IN: ED7006C1 ED7006D1 CD7006E
NEW_MEM_SIZE 16_777_216

-- $LOW_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE LOWER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN: CD7007C
LOW_PRIORITY 0

-- $HIGH_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE UPPER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN: CD7007C
HIGH_PRIORITY 99

-- $MANTISSA_DOC
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_MANTISSA AS SPECIFIED
-- IN THE IMPLEMENTOR'S DOCUMENTATION.
-- USED IN: CD7013B
MANTISSA_DOC 31

-- $DELTA_DOC
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.FINE_DELTA AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN: CD7013D
DELTA_DOC 2.0**(-31)

-- $TICK
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.TICK AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN: CD7104B
TICK 0.01666666

```

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G

This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E

This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A

This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B

This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D

This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]

These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110

These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C

These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B

This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B

This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]

These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A

This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B

These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D

This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I

This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C

This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A

This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B

This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to lack of available disk space, and the test would thus encumber validation testing.

E28005C

This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

Computer Sciences Corporation

Compiler: MC Ada V1.2.beta/ Concurrent Computer Corporation

ACVC Version: 1.10

Ada Compiler Command Directives

Explicit options used with the Ada command during the validation:

- E to generate an error file
- w to suppress warnings(except for test e28002a)

Default options used with the Ada command during validation:

- O0 Default compiler optimization setting

Ada Library Directives

The LIBRARY:LINK and TASKING:LINK directives in ada.lib was changed to point to the ARTOS unique library. This library provides the necessary binding procedures to the ARTOS.

LIBRARY:LINK:/acvc10/artos.lib:
TASKING:LINK:/acvc10/artos.lib:

The INFO directive, specified in the ada.lib file, is used to control the type of floating point instructions generated.

The following directive generates floating point instructions that are executed on the MC68881 co-processor. This was used during the Ada validation.

FLOATING_POINT_SUPPORT:INFO:MC68881:

Ada Linker Directives

The following options were used with the ada linker:

- o executable_unit_name to name and direct the executable file to a specific directory: /acvc10/out.
executable_unit_name is the ACVC V1.10 procedure name.
- z to force linker to align text, data, and BSS domains along page boundary. A page is 4096 bytes. This option is actually an option of the Masscomp RTU supplied linker. The Ada Linker shell simply passes it to the linker.
- s to suppress generation of symbol table in executable file and conserve disk space.

Example:

```
a.ld -o /acvc10/out/c23001a.out c23001a -z -s
```

ARTOS options:

The ARTOS operating system is booted while at the MASSCOMP 5600 console mode as follows:

```
>>>boot /rte/artos
```

ARTOS is set up to automatically enter the Debug mode. While at the debug mode ARTOS may be commanded to execute user programs with a specific processor. This option is exercised to insure that all executable tests pass under both a single processor and a multi-processing environment. First all tests were forced to execute on processor 1 by typing the following directive:

```
ARTOS.Debug:=run 2
```

where 2 is the bit mask for processor 1.

Then, to insure proper multi-processing operation, the tests were also executed using both processors by typing the following directive:

```
ARTOS.Debug:=run 6
```

where 6 is the bit mask for processor 1 and 2.

Ada Compiler Command and Options

SYNTAX:

ada [options] [ada_source.a]...[ld_options] [object_file.o]...

DESCRIPTION:

The command `ada` executes the Ada compiler and compiles the named Ada source file, ending with the `.a` suffix. The file must reside in a VADS library directory. If the source file name has the form `aaa/bbb.a`, `aaa` must be a VADS library directory. The `ada.lib` file in this directory is modified after each Ada unit is compiled.

All files created and modified are located in subdirectories of the directory that contains the `ada_source.a` file. The object program is left in a file with the same name as that of the source with `.o` substituted for `.a` unless the `-o` option is used. The object file is written to the `.objects` subdirectory of the VADS library.

By default, `ada` produces only object and net files. If the `-M` option is used, the compiler automatically invokes `a.ld` and builds a complete program with the named library unit as the main program.

Non-Ada object files (e.g., `.o` files produced by the C or FORTRAN compiler) may be given as arguments to `ada`. These files will be passed on to the linker and will be linked with the specified Ada object files.

Command line options may be specified in any order, but the order of compilation and the order of the files to be passed to the linker can be significant.

OPTIONS:

- `-a file_name` (archive) Treat `file_name` as an ar file. Since archive files end with `.a`, `-a` is used to distinguish archive files from Ada source files.
- `-d` (dependencies) Analyze for dependencies only. Do not do semantic analysis or code generation. Update the library, marking any defined units as uncompiled. The `-d` option is used by `a.make` to establish dependencies among new files.

-e (error) Process compilation error messages using a.error and direct it to stdout. Only one -e or -E option should be used.

-E
 -E file
 -E directory (error output) Without a file or directory argument, ada processes error messages using a.error and directs the output to stdout; the raw error messages are left in source.err. If a file pathname is given, the raw error messages are placed in that file. If a directory argument is supplied, the raw error output is placed in dir/source.err. Only one -e or -E option should be used.

-el (error listing) Intersperse error messages among source lines and direct to stdout.

-El
 -El file
 -El directory (error listing) Same as the -E option, except that source listing with errors is produced.

-ev (error vi) Process the raw error messages using a.error, embed them in the source file, and call vi on the source file.

-lfile_abbreviation (link) Link this library file. (Do not space between the -l and the file abbreviation.) See ld(1).

-M unit_name (main) Produce an executable program using the named unit as the main program. The unit must be either a parameterless procedure or a parameterless function returning an integer. The executable program will be left in the file a.out unless overridden with the -o option.

-M ada_source.a (main) Like -M unit_name, except that the unit name is assumed to be the root name of the .a file (e.g., for foo.a the unit is foo). Only one .a file may be preceded by -M.

-o executable file (output) This option is to be used in conjunction with the -M option.

executable_file is the name of the executable rather than the default a.out.

- O[1-9] (optimize) Invoke the code optimizer. An optional digit limits the number of optimization passes. The default, 0, optimizes as far as possible.
- R VADS_library (recompile instantiation) Force analysis of all generic instantiations, causing reinstatement of any that are out of date.
- S (suppress) Apply pragma SUPPRESS to the entire compilation.
- T (timing) Print timing information for the compilation.
- u (update) Cause library status to reflect the current program source. Unless the source is syntactically incorrect, the compiler updates the library ada.lib. Normally, the library is changed only if the unit compiles without errors of any kind.
- v (verbose) Print compiler version number, date and time of compilation, name of file compiled, command input line, total compilation time, and error summary line.
- w (warnings) Suppress warning diagnostics.

FILES:

file.a Ada source input file
/tmp/file.\$\$ IL code file created by front end
ada.lib VADS directory information file
gnrx.lib VADS generics library information file
GVAS_table GVAS table in the current VADS project
ada.lock lock link to ada.lib, for mutual exclusion
gnrx.lock lock generics library, for mutual exclusion
GVAS_table.LOCK
lock link to GVAS_table, for mutual exclusion

SEE ALSO:

a.db, a.error, a.ld, a.mklib, ld(1)

DIAGNOSTICS:

The diagnostics by the VADS compiler are intended to be self-explanatory. Most refer to the RM. Each RM reference includes a section number and optionally, a paragraph number enclosed in parentheses.

Ada Linker Command and Options

SYNTAX:

a.ld [options] unit_name [ld_options]

DESCRIPTION:

a.ld collects the object files needed to make unit_name a main program and calls the UNIX linker ld(1) to link together all Ada and other language objects required to produce an executable image in a.out. unit_name is the main program and must be a non-generic subprogram. If unit_name is a function, it must return a value of the type STANDARD.INTEGER. This integer result will be passed back to the UNIX shell as the status code of the execution. The utility uses the net files produced by the Ada compiler to check dependency information. a.ld produces an exception mapping table and a unit elaboration table and passes this information to the linker.

a.ld reads instructions for generating executables from the ada.lib file in the VADS libraries on the search list. Besides information generated by the compiler, these directives also include WITH_n directives that allow the automatic linking of object modules compiled from other languages or Ada object modules not named in context clauses in the Ada source. Any number of WITH directives may be placed into a library, but they must be numbered contiguously beginning at WITH1. The directives are recorded in the library's ada.lib file and have the following form.

WITH1:LINK:object_file:
WITH2:LINK:archive_file:

WITH directives may be placed in the local Ada libraries or in any VADS library on the search list.

A WITH directive in a local VADS library or earlier on the library search list will hide the same numbered WITH directive in a library later in the library search list.

Use the tool a.info to change or report library directives in the current library.

All arguments after unit_name are passed on to the linker. These may be options for it, archive libraries, library abbreviations, or object files.

VADS_location/bin/a.ld is a wrapper program that executes the correct executable based upon directives visible in the ada.lib file. This permits multiple VADS compilers to exist on the same host. The sh option prints the name of the actual executable file.

OPTIONS:

- E unit_name (elaborate) Elaborate unit_name as early in the elaboration order as possible.
- F (files) Print a list of dependent files in order and suppress linking.
- o executable_file (output) Use the specified file name the name of the output rather than the default, a.out.
- sh (show) Display the name of the tool executable but do not execute it.
- U (units) Print a list of dependent units in order and suppress linking.
- v (verbose) Print the linker command before executing it.
- V (verify) Print the linker command but suppress execution.

FILES:

VADS_location/standard/* startup and standard library routines
Ada object files
a.out default output file

SEE ALSO:

Operating system_documentation, ld(1)

DIAGNOSTICS:

Self-explanatory diagnostics are produced for missing files, etc. occasional additional messages are produced by the linker.

ARTOS Commands and Options

At the debug level:

run <processor_mask>

processor_mask is an hexadecimal value that specifies the processors that are to be used to execute user programs. A one at the processor's bit position indicates that it can run user's processes. Processors bit assignment is 0 .. 7 corresponding to bit position 0 .. 7 from right to left.

At the command interpreter level:

echo <message_string>

this directive causes the message_string to be output to the standard output device.

change <directory_path>

this directive causes the default directory to be changed to the directory path specified. directory names are separated by '/'. If directory_path contains a leading '/', then it is relative to the root directory. Otherwise, it is assumed it is relative to the current directory. It is assumed the directory and file structure to be UNIX compatible.

shell <[path/]text_file>

this directive causes the command contained within the specified text_file to be executed.

return

this directive indicates the 'end_of_command' within the text_file and it causes the previous command level to be returned to.

run <[path/]executable_file>

this directive causes the specified executable_file to be loaded and executed as a new process. The current process wait for the new process to complete before reading the next directive.

NOTE: Only the commands/options used in the execution of the
ACVC tests are described.